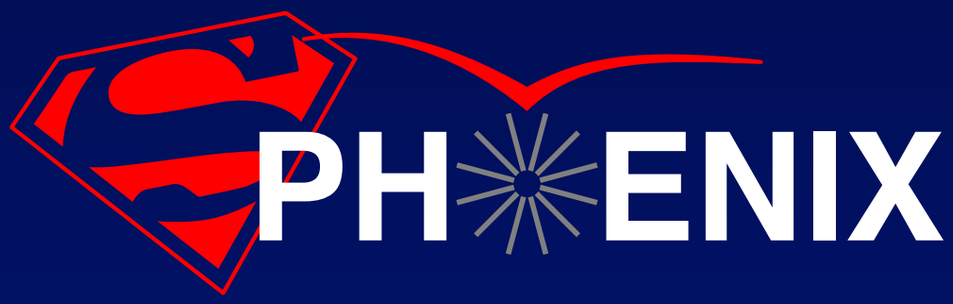
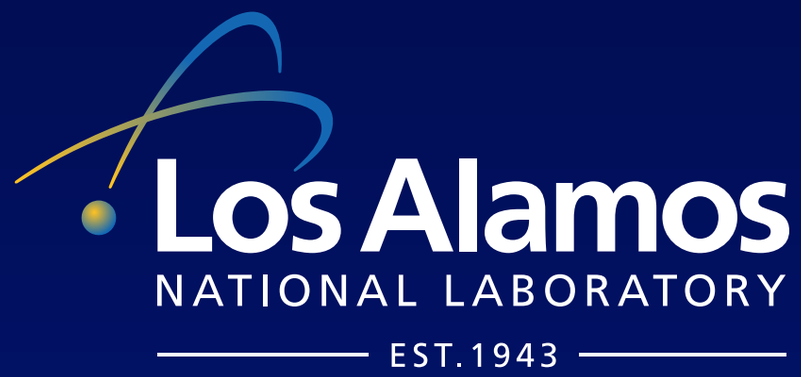
Pull Request #101

Lossy Calorimeter Truth Compression via PHG4Shower

Michael P. McCumber
Los Alamos National Laboratory

sPHENIX Simulations Meeting
1/5/2016





Old Truth Ancestry

OLD TRUTH ANCESTRY

PHG4Particle

Old Truth Ancestry

Old Truth Ancestry

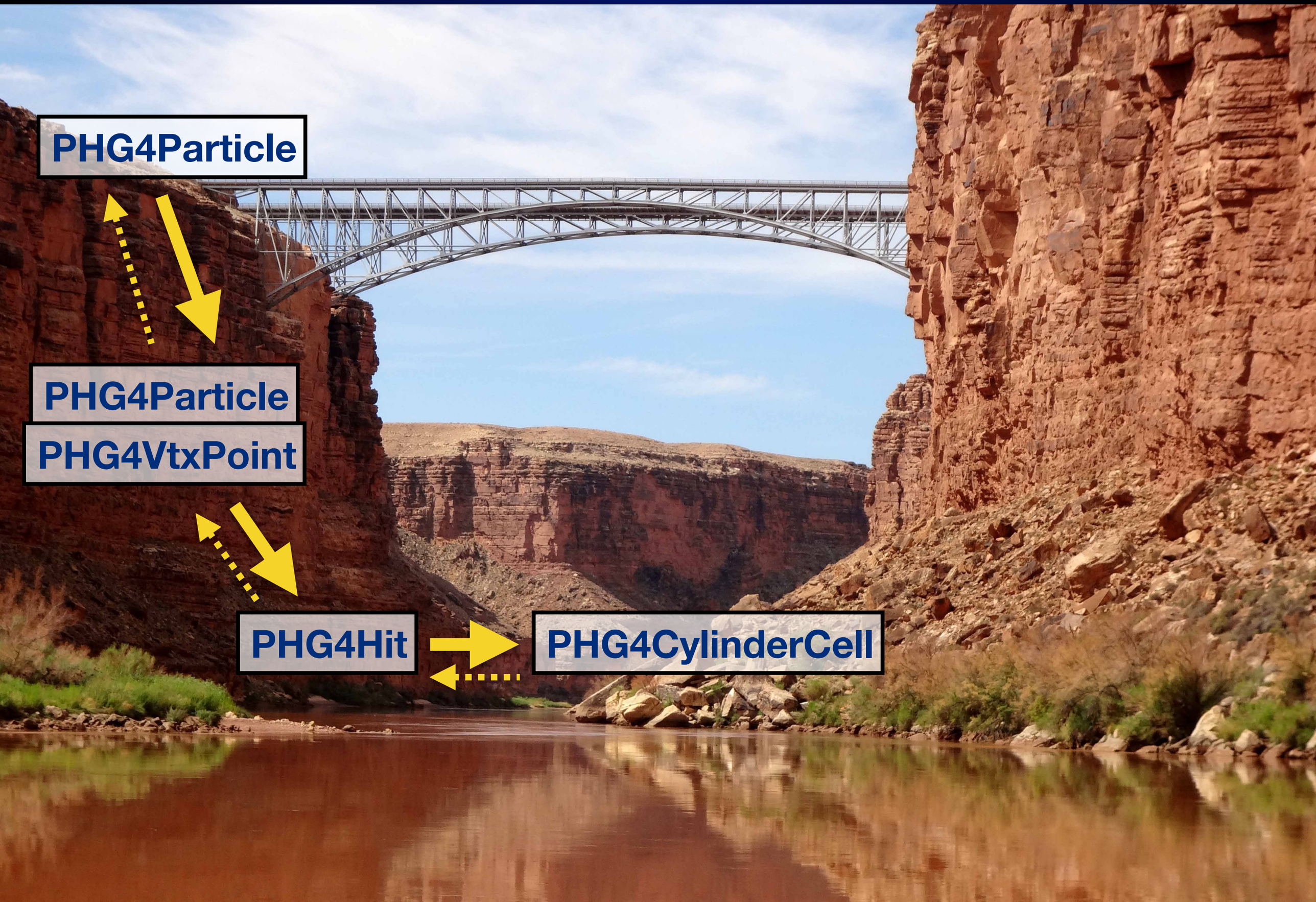
PHG4Particle

PHG4Particle

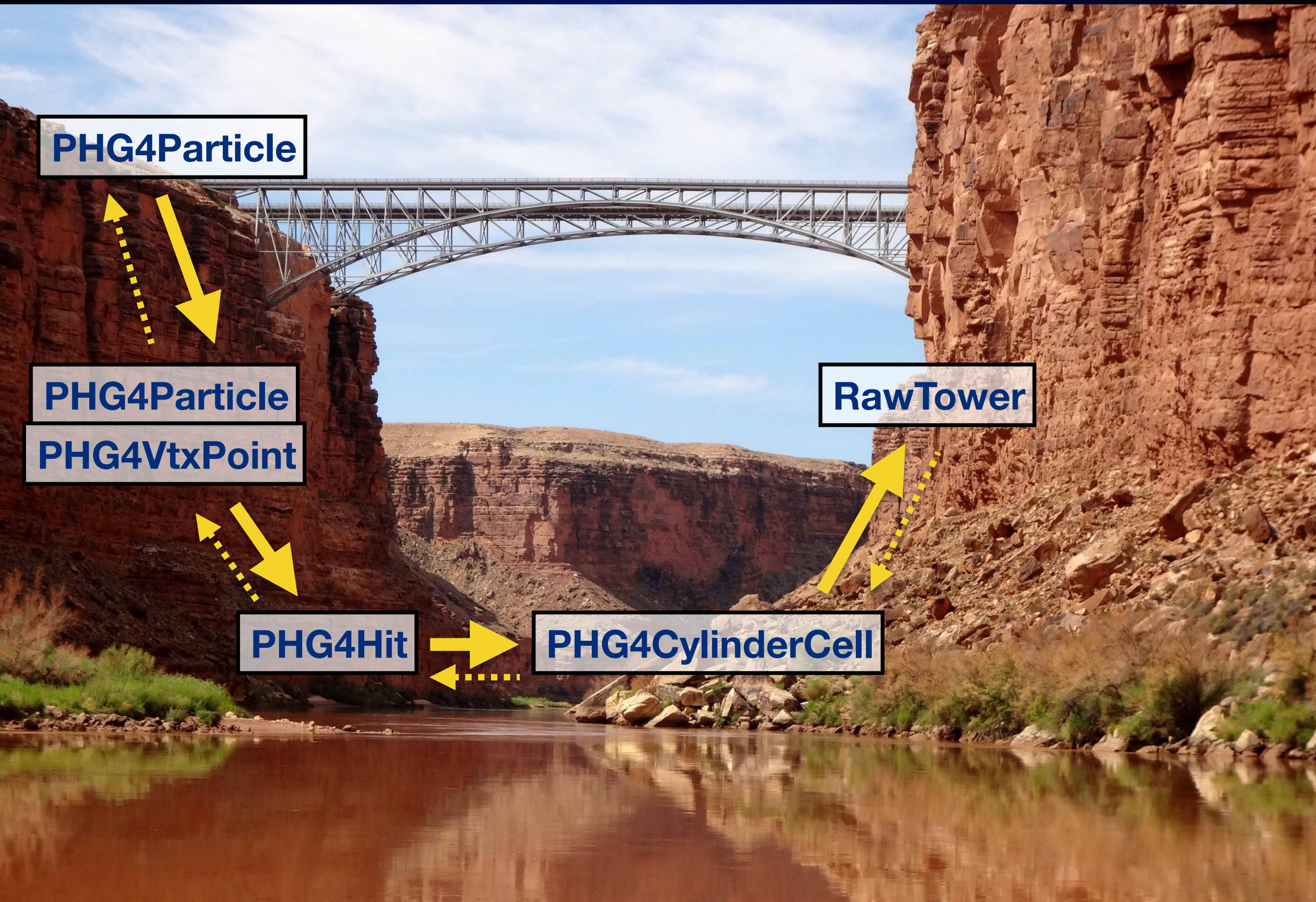
PHG4VtxPoint

PHG4Hit

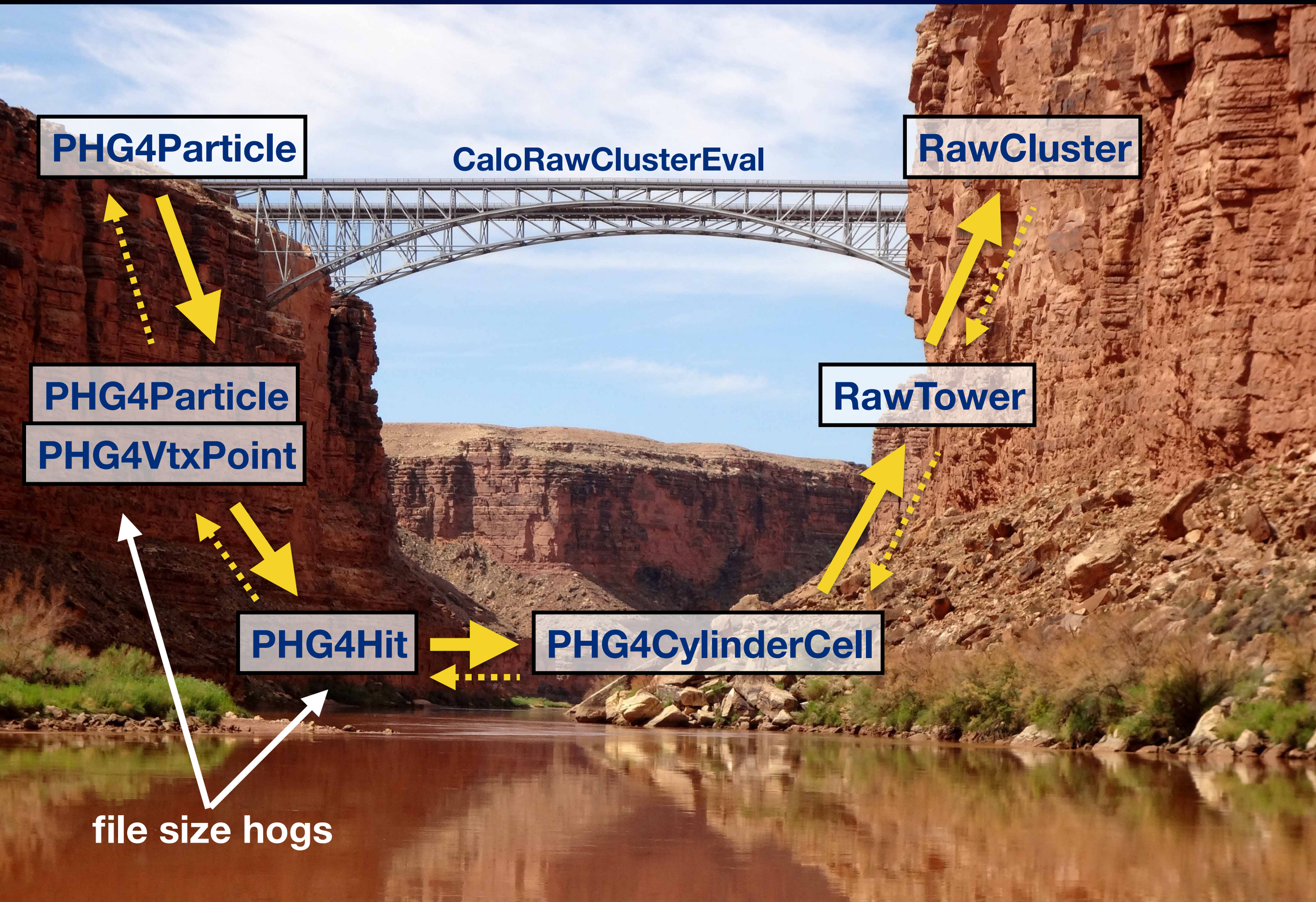
OLD TRUTH ANCESTRY



OLD TRUTH ANCESTRY

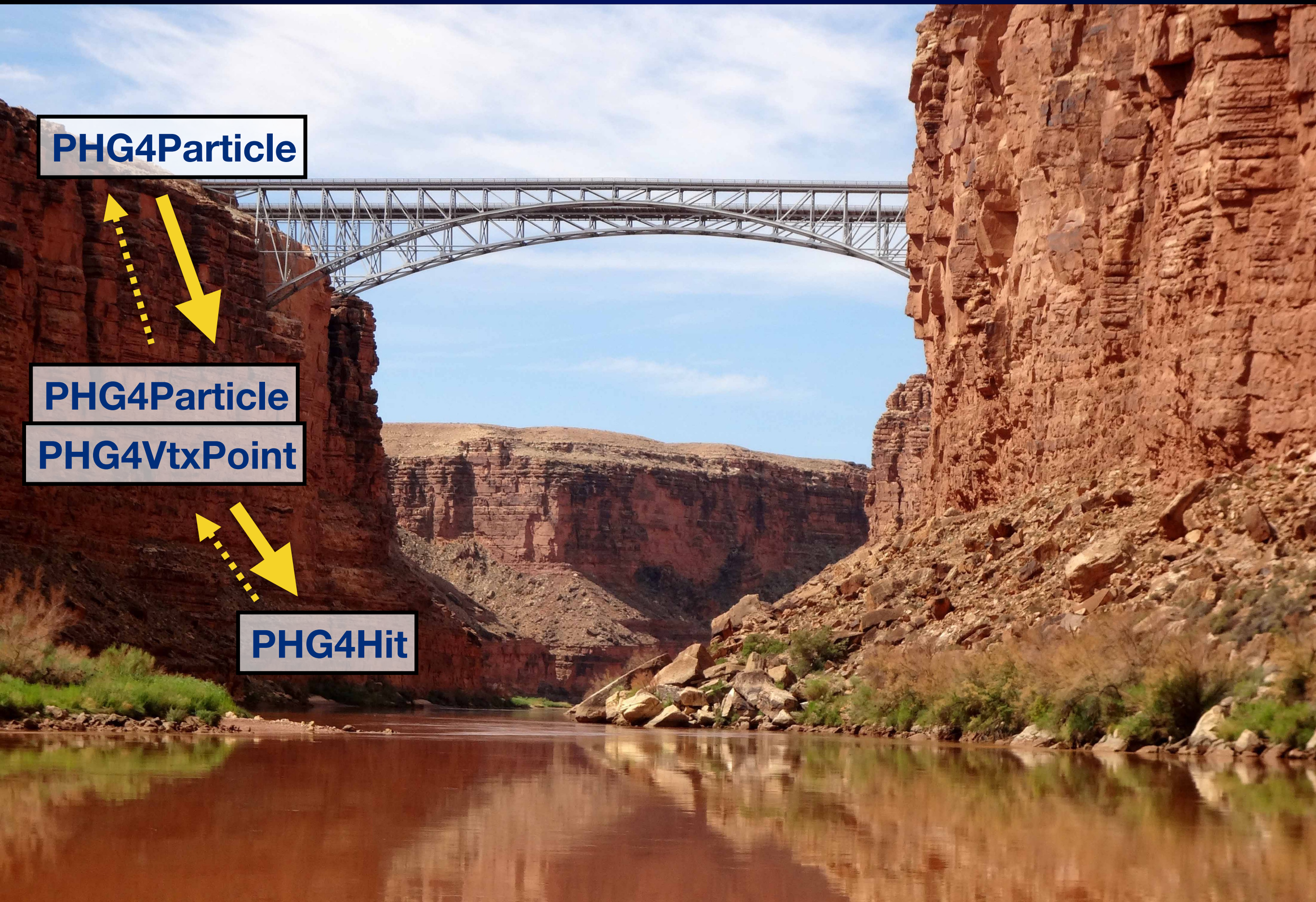


Old Truth Ancestry

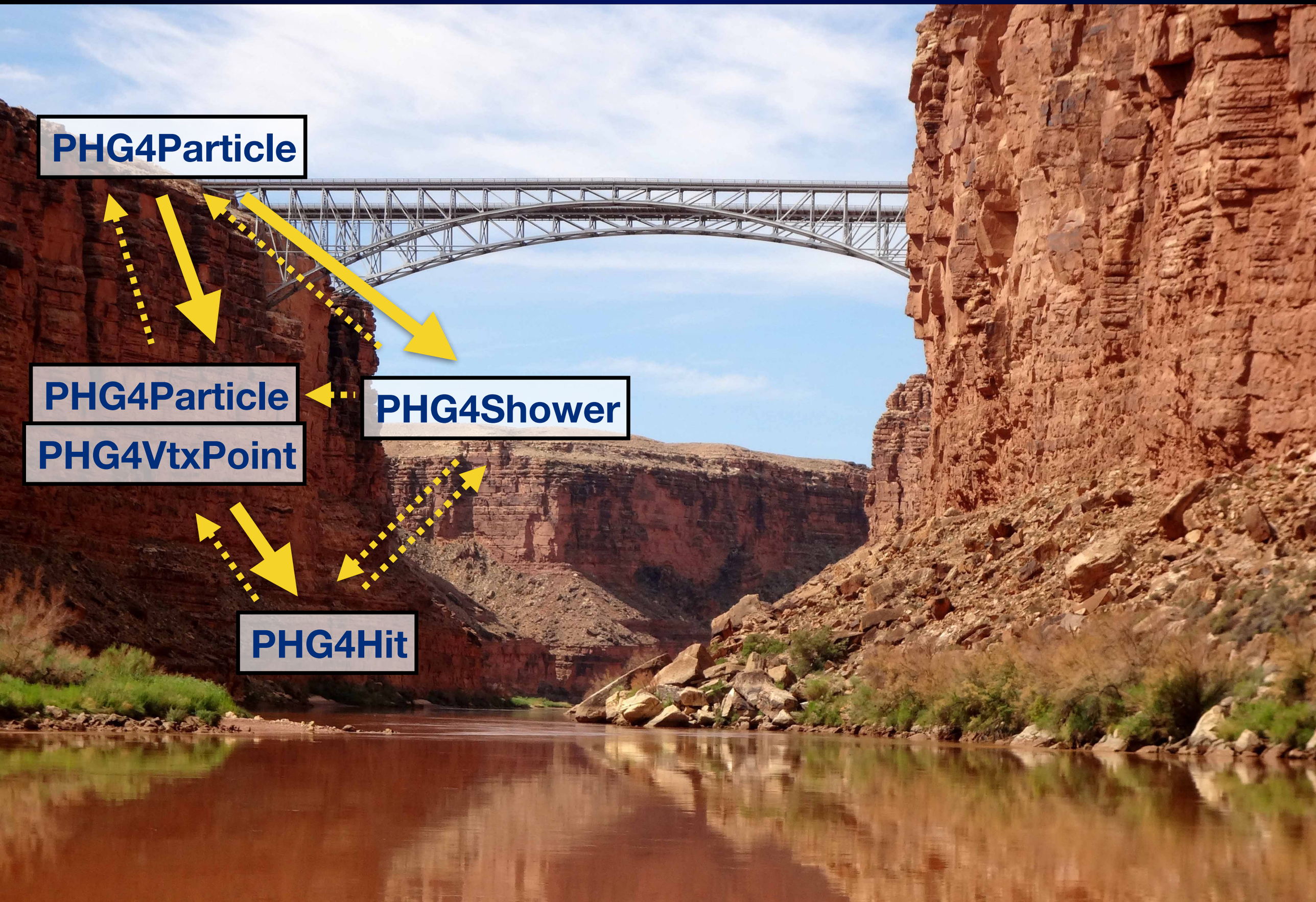


New Truth Ancestry

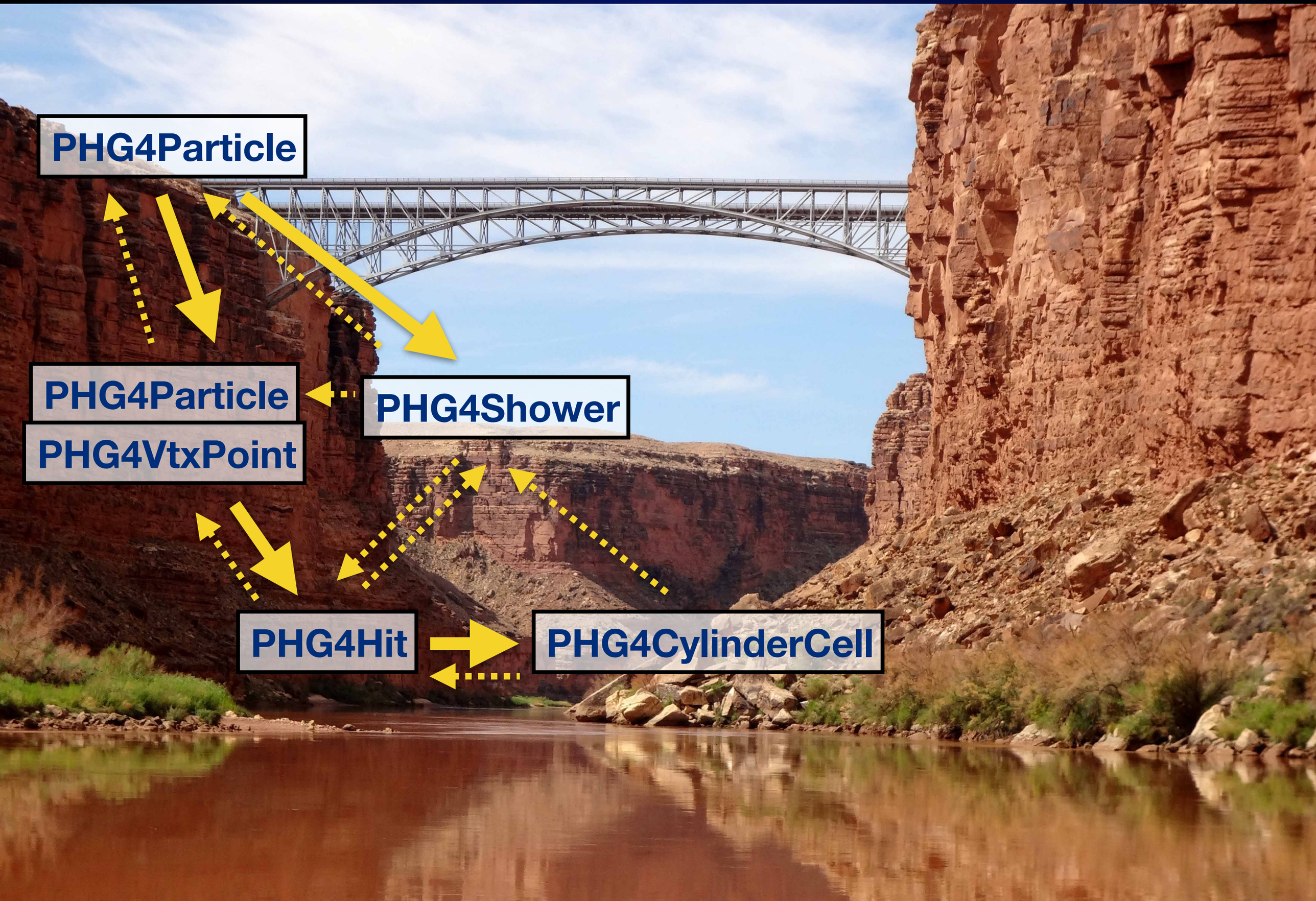
New Truth Ancestry



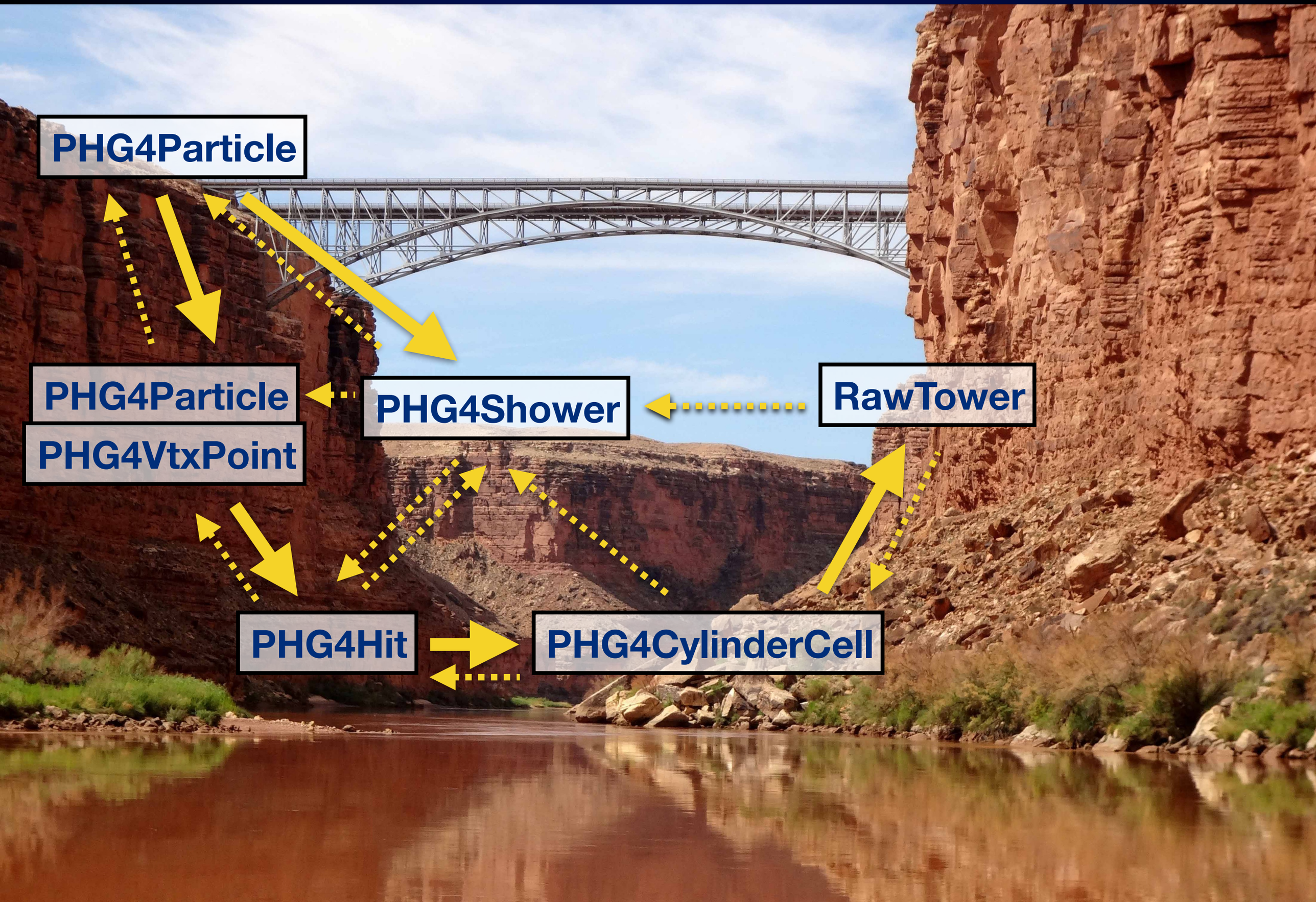
New Truth Ancestry



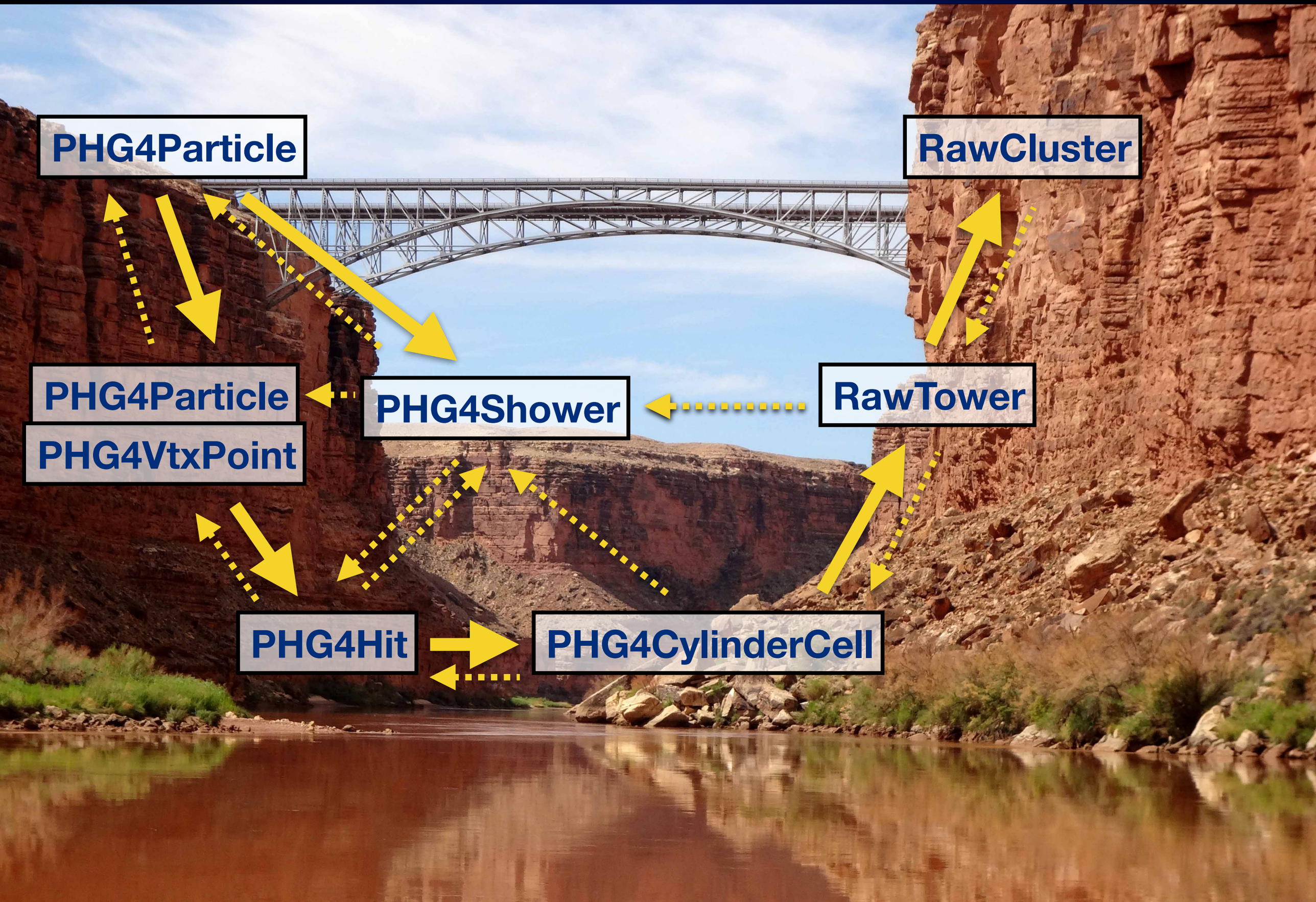
New Truth Ancestry



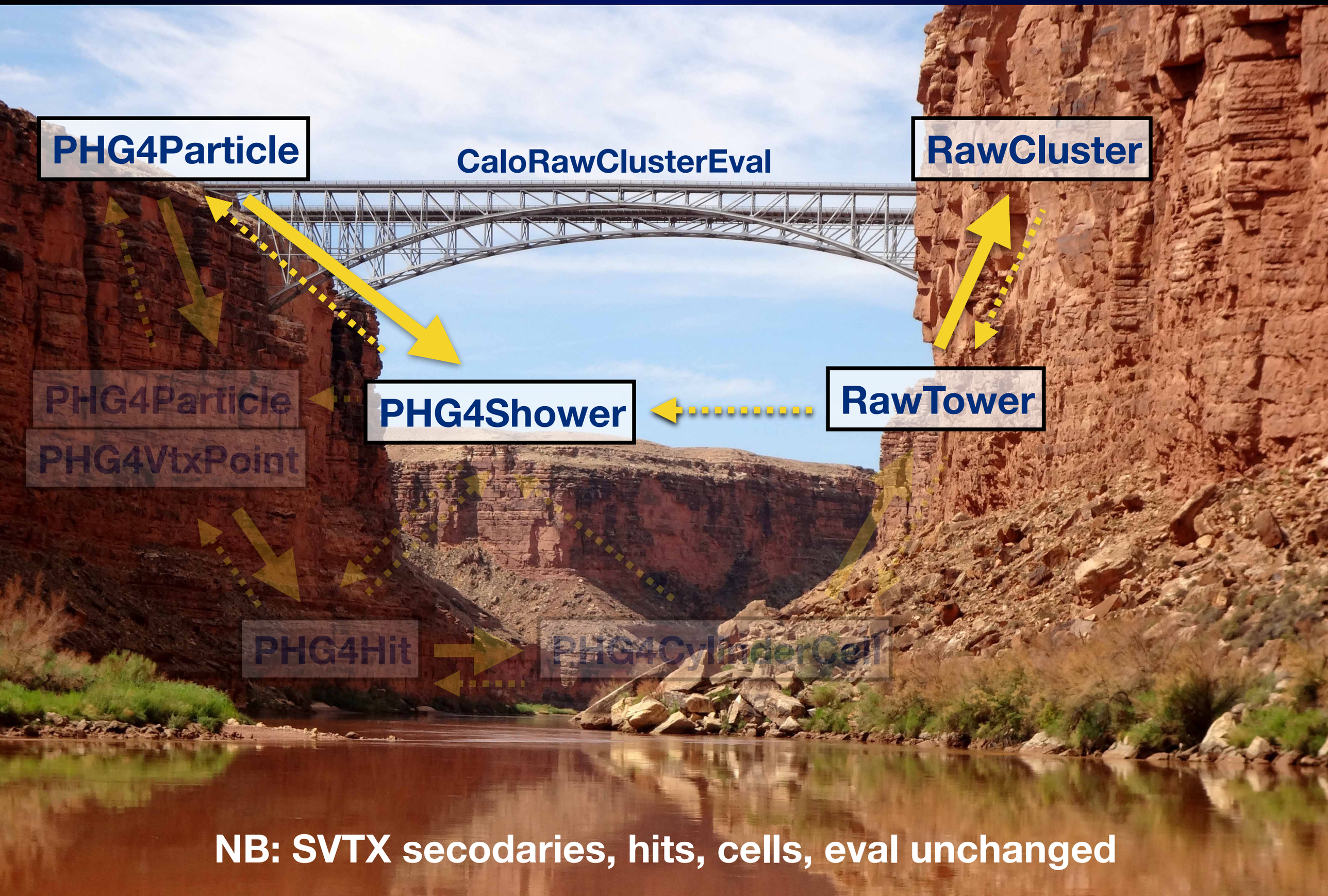
New Truth Ancestry



New Truth Ancestry



New Truth Ancestry



NB: SVTX secodaries, hits, cells, eval unchanged

PHG4Shower Object

```
int                 _id;                    //< unique identifier within container
int                 _primary_particle_id; //< association of shower to primary particle id
int                 _parent_particle_id;  //< association of shower to parent particle id
int                 _primary_shower_id;   //< association of shower to primary shower id
int                 _parent_shower_id;    //< association of shower to parent shower id
float               _pos[3];              //< mean position of the shower
float               _covar[6];            //< covariance of shower positions
std::map<int, unsigned int> _nhits;       //< number of hits in different volumes
std::map<int, float> _edep;               //< energy deposit in different volumes
std::map<int, float> _eion;               //< ionization energy in different volumes
std::map<int, float> _light_yield;        //< light yield in different volumes
std::map<int, float> _moliere_radius;     //< moliere radius in different volumes
std::map<int, float> _eh_ratio;           //< electron/hadron ratio of energy in different volumes

std::set<int> _g4particle_ids;
std::set<int> _g4vertex_ids;
std::map<int,std::set<unsigned long long> > _g4hit_ids;
```

Showers are stored in PHG4TruthInfoContainer:

```
//! Get a range of iterators covering the entire container
ShowerRange GetShowerRange() {return ShowerRange(showermap.begin(),showermap.end());}
ConstShowerRange GetShowerRange() const {return ConstShowerRange(showermap.begin(),showermap.end());}
```

SHOWER CREATION

Shower objects are automatically created for **all volumes and all PHG4HitContainers**

PHG4TruthInfoTrackingAction::PreUserTrackingAction()

```
109
110     // create or add to a new shower object ------------------------------------
111     if (!track->GetParentID()) {
112       PHG4Showerv1* shower = new PHG4Showerv1();
113       PHG4TrackUserInfo::SetShower(const_cast<G4Track *> (track), shower);
114       truthInfoList_->AddShower(trackid, shower);
115       shower->set_id(trackid);
116       shower->set_primary_particle_id(trackid);
117       shower->set_primary_shower_id(trackid);
118       shower->set_parent_particle_id(0);
119       shower->set_parent_shower_id(0);
120     } else {
121       // get shower
122       if ( G4VUserTrackInformation* p = track->GetUserInformation() ) {
123         if ( PHG4TrackUserInfoV1* pp = dynamic_cast<PHG4TrackUserInfoV1*>(p) ) {
124           if (pp->GetShower()) {
125             pp->GetShower()->add_g4particle_id(trackid);
126             pp->GetShower()->add_g4vertex_id(vtxindex);
127           }
128         }
129       }
130     }
```

each primary input particle gets a shower object

SHOWER CREATION #2

PHG4TruthInfoTrackingAction::PostUserTrackingAction()

```
148     int trackid = track->GetTrackID();
149     int primaryid = 0;
150     PHG4Shower* shower = NULL;
151     if ( PHG4TrackUserInfoV1* p = dynamic_cast<PHG4TrackUserInfoV1*>(track->GetUserInformation()) ) {
152       trackid = p->GetUserTrackId();
153       primaryid = p->GetUserPrimaryId();
154       shower = p->GetShower();
155     }
156
157     G4TrackVector* secondaries = fpTrackingManager->GimmeSecondaries();
158     if (secondaries) {
159       for (size_t i = 0; i < secondaries->size(); ++i) {
160         G4Track* secondary = (*secondaries)[i];
161         PHG4TrackUserInfo::SetUserParentId(const_cast<G4Track *> (secondary), trackid);
162         PHG4TrackUserInfo::SetUserPrimaryId(const_cast<G4Track *> (secondary), primaryid);
163         PHG4TrackUserInfo::SetShower(const_cast<G4Track *> (secondary), shower);
164       }
165     }
166   }
```

each secondary created is passed a pointer to the primary shower
(neat trick for passing information forward in GEANT)

SHOWER CREATION #3

PHG4*SteppingAction::SteppingAction()

```
92          //set the track ID
93          {
94            hit->set_trkid(aTrack->GetTrackID());
95            if ( G4VUserTrackInformation* p = aTrack->GetUserInformation() )
96              {
97                if ( PHG4TrackUserInfoV1* pp = dynamic_cast<PHG4TrackUserInfoV1*>(p) )
98                  {
99                    hit->set_trkid(pp->GetUserTrackId());
100                   hit->set_shower_id(pp->GetShower()->get_id());
101                 }
```

hits are told which shower they belong to

```
92          //set the track ID
93          {
94            hit->set_trkid(aTrack->GetTrackID());
95            if ( G4VUserTrackInformation* p = aTrack->GetUserInformation() )
96              {
97                if ( PHG4TrackUserInfoV1* pp = dynamic_cast<PHG4TrackUserInfoV1*>(p) )
98                  {
99                    hit->set_trkid(pp->GetUserTrackId());
100                   hit->set_shower_id(pp->GetShower()->get_id());
101                 }
```

showers are told which hits they contain

SHOWER CREATION #4

PHG4TruthEventAction::EndOfEventAction(),PruneShowers(), ProcessShowers()

```
497
498       // mean value of shower
499       double prefactor = 1.0 / sumw;
500       Eigen::Matrix<double, 1, 3> mean = prefactor * W.transpose() * X;
501
502       // compute residual relative to the mean
503       for (unsigned int i=0; i<points.size(); ++i) {
504         for (unsigned int j=0; j<3; ++j) X(i,j) = points[i][j] - mean(0,j);
505       }
506
507       // weighted covariance matrix
508       prefactor = sumw / (pow(sumw,2) - sumw2); // effectivelly 1/(N-1) when w_i = 1.0
509       Eigen::Matrix<double, 3, 3> covar = prefactor * (X.transpose() * W.asDiagonal() * X);
510
511       shower->set_x(mean(0,0));
512       shower->set_y(mean(0,1));
513       shower->set_z(mean(0,2));
514
515       for (unsigned int i = 0; i < 3; ++i) {
516         for (unsigned int j = 0; j <= i; ++j) {
517           shower->set_covar(i,j,covar(i,j));
518         }
519       }
```

Showers are cycled to remove zero energy hits, and summarize characteristics (example above shows energy weighted position PCA analysis)

RawTower Modification

RawTowerv1.h

dual interface functions for cells and showers (similar set of mods to Cell objects)

```
32
33    //---cell access-----------------------------------------------------------
34
35    bool empty_g4cells() const { return ecells.empty(); }
36    size_t size_g4cells() const { return ecells.size(); }
37    RawTower::CellConstRange get_g4cells() const {
38      return make_pair(ecells.begin(), ecells.end());
39    }
40    RawTower::CellIterator find_g4cell(int id) { return ecells.find(id); }
41    RawTower::CellConstIterator find_g4cell(int id) const {return ecells.find(id);}
42    void add_ecell(const PHG4CylinderCellDefs::keytype g4cellid,
43                   const float ecell);
44    void clear_g4cells() { ecells.clear(); }
45
46    //---shower access---------------------------------------------------------
47
48    bool empty_g4showers() const { return eshowers.empty(); }
49    size_t size_g4showers() const { return eshowers.size(); }
50    RawTower::ShowerConstRange get_g4showers() const {
51      return make_pair(eshowers.begin(), eshowers.end());
52    }
53    RawTower::ShowerIterator find_g4shower(int id) { return eshowers.find(id); }
54    RawTower::ShowerConstIterator find_g4shower(int id) const {return eshowers.find(id);}
55    void add_eshower(const int g4showerid, const float eshower);
56    void clear_g4showers() { eshowers.clear(); }
57
```

TYPICAL USER EVAL OBJECT CHANGES

CaloRawClusterEval.h

```
// ---reduced sim node or better--------------------------------------------

/// has the eval initialized correctly for reduced sim DST nodes?
bool has_reduced_node_pointers();

// shower interface

/// what primary showers contributed energy to this cluster?
std::set<PHG4Shower*> all_truth_primary_showers (RawCluster* cluster);

/// which primary shower contributed the most energy to this cluster?
PHG4Shower* max_truth_primary_shower_by_energy (RawCluster* cluster);

/// what clusters did this primary truth shower contribute energy to?
std::set<RawCluster*> all_clusters_from(PHG4Shower* primary);

/// which cluster did this primary truth shower contribute the most energy to?
RawCluster* best_cluster_from(PHG4Shower* primary);

/// how much energy did this primary truth shower contribute to this cluster
float get_energy_contribution (RawCluster* cluster, PHG4Shower* primary);

// particle interface

/// what particles contributed energy to this cluster?
std::set<PHG4Particle*> all_truth_primary_particles (RawCluster* cluster);

/// which particle contributed the most energy to this cluster?
PHG4Particle* max_truth_primary_particle_by_energy (RawCluster* cluster);

/// what clusters did this primary truth particle contribute energy to?
std::set<RawCluster*> all_clusters_from(PHG4Particle* primary);

/// which cluster did this primary truth particle contribute the most energy to?
RawCluster* best_cluster_from(PHG4Particle* primary);

/// how much energy did this primary truth particle contribute to this cluster
float get_energy_contribution (RawCluster* cluster, PHG4Particle* primary);
```

works with reduced DSTs

works with reduced DSTs or full DST (needs PHG4Hits)

```
// ---full sim node required------------------------------------

/// has the eval initialized correctly for full sim DST nodes?
bool has_full_node_pointers();

/// what truth hits contributed energy to this tower?
std::set<PHG4Hit*> all_truth_hits (RawCluster* cluster);
```

cluster<=>particle interface unchanged, but uses shower storage for CPU benefit

NB: there minor name changes to break shower/particle naming collisions

Reduction Of DST

Shower objects are automatically created for all volumes and all PHG4HitContainers
but deletions are done manually for specified containers

```
PHG4DstCompressReco* compress = new PHG4DstCompressReco("PHG4DstCompressReco");
compress->AddHitContainer("G4HIT_CEMC_ELECTRONICS");
compress->AddHitContainer("G4HIT_CEMC");
compress->AddHitContainer("G4HIT_ABSORBER_CEMC");
compress->AddHitContainer("G4HIT_CEMC_SPT");
compress->AddHitContainer("G4HIT_ABSORBER_HCALIN");
compress->AddHitContainer("G4HIT_HCALIN");
compress->AddHitContainer("G4HIT_HCALIN_SPT");
compress->AddHitContainer("G4HIT_MAGNET");
compress->AddHitContainer("G4HIT_ABSORBER_HCALOUT");
compress->AddHitContainer("G4HIT_HCALOUT");
compress->AddHitContainer("G4HIT_BH_1");
compress->AddHitContainer("G4HIT_BH_FORWARD_PLUS");
compress->AddHitContainer("G4HIT_BH_FORWARD_NEG");
compress->AddCellContainer("G4CELL_CEMC");
compress->AddCellContainer("G4CELL_HCALIN");
compress->AddCellContainer("G4CELL_HCALOUT");
compress->AddTowerContainer("TOWER_SIM_CEMC");
compress->AddTowerContainer("TOWER_RAW_CEMC");
compress->AddTowerContainer("TOWER_CALIB_CEMC");
compress->AddTowerContainer("TOWER_SIM_HCALIN");
compress->AddTowerContainer("TOWER_RAW_HCALIN");
compress->AddTowerContainer("TOWER_CALIB_HCALIN");
compress->AddTowerContainer("TOWER_SIM_HCALOUT");
compress->AddTowerContainer("TOWER_RAW_HCALOUT");
compress->AddTowerContainer("TOWER_CALIB_HCALOUT");
se->registerSubsystem(compress);
```

So with this list, showers are created for the forward calorimeters and utilized by the evaluation, but not yet deleted.

Secondary particles and vertexes are preserved by sweeping through all hit containers named “G4HIT_*” and preserving objects needed by non-designated objects. Thus, a forward tracker when implemented will preserve its truth information

DST SIZE PERFORMANCE

5 random Central HIJING 4-7 fm event with absorber hits

```
[mccumber@geant4 ~/sphenix]> ls -lSh | head -n 5
total 642M
-r--r--r-- 1 mccumber rhphenix 621M Jan  4 00:50 G4sPHENIXCells_fullsize.root
-r--r--r-- 1 mccumber rhphenix  17M Jan  4 13:43 G4sPHENIXCells_reducedsize.root
-rw-r--r-- 1 mccumber rhphenix 3.2M Jan  3 19:36 g4svtx_eval.root
```

Without compression: 621MB / 5 events = 124 MB / event

With shower compression: 17 MB / 5 events = 3.4 MB / event

3.4 MB/event, a factor 37 better than full DST

What dominates the DST file size after compression?

19% SVTXSUPPORT G4Hits <= add to compression

16% SVTX G4Hits

14% PIPE G4Hits <= add this to compression

13% PHG4Showers

11% SimTowers

10% PHG4Particles <= includes SVTXSUPPORT/PIPE secondaries

7% GenEvent record

6% Raw and Calib towers <= these could be regenerated

4% PHG4Vertexes <= includes SVTXSUPPORT/PIPE secondaries

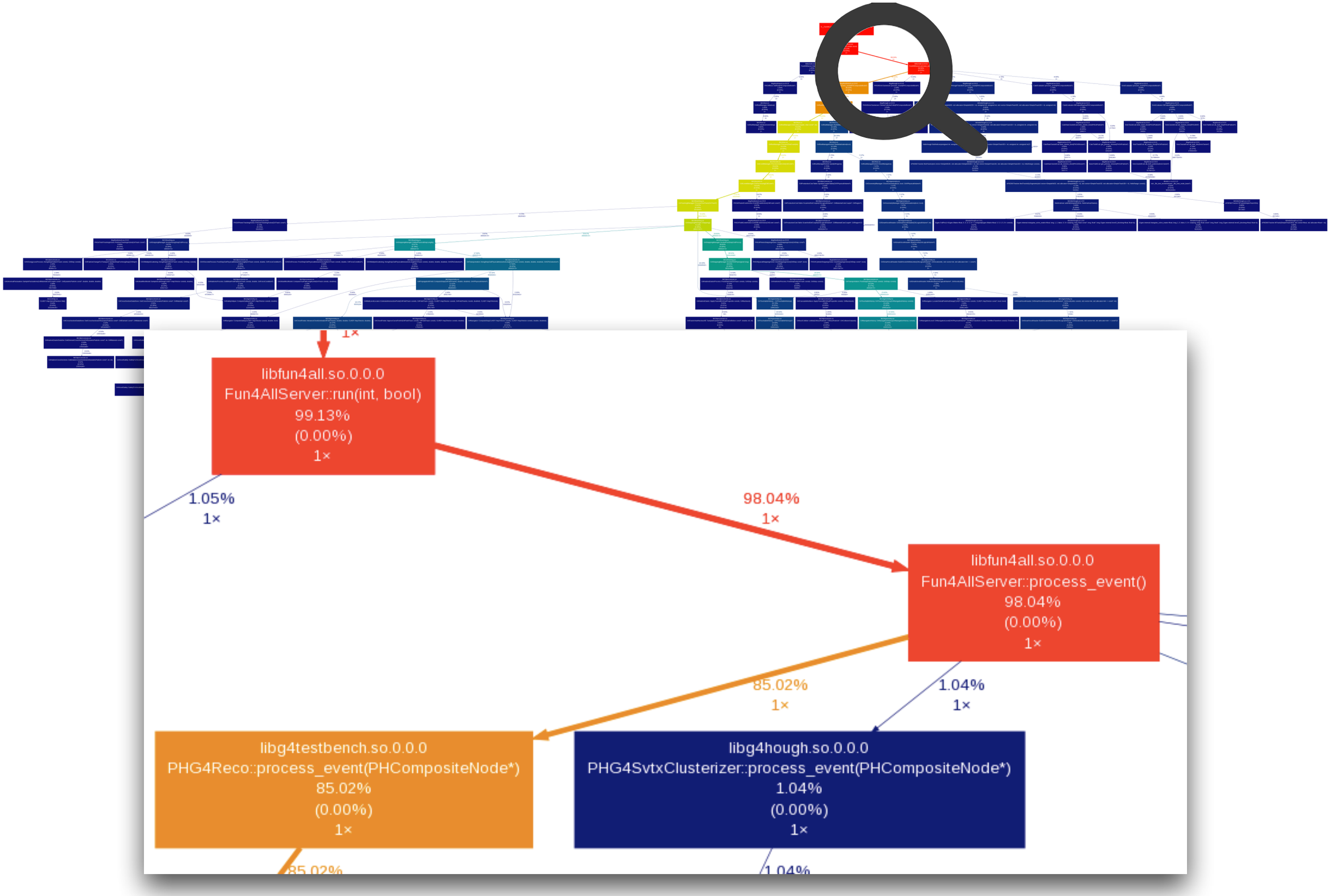
A factor ~70 might be possible with some additional cleanup

PROFILING MID-CENTRAL EVENTS

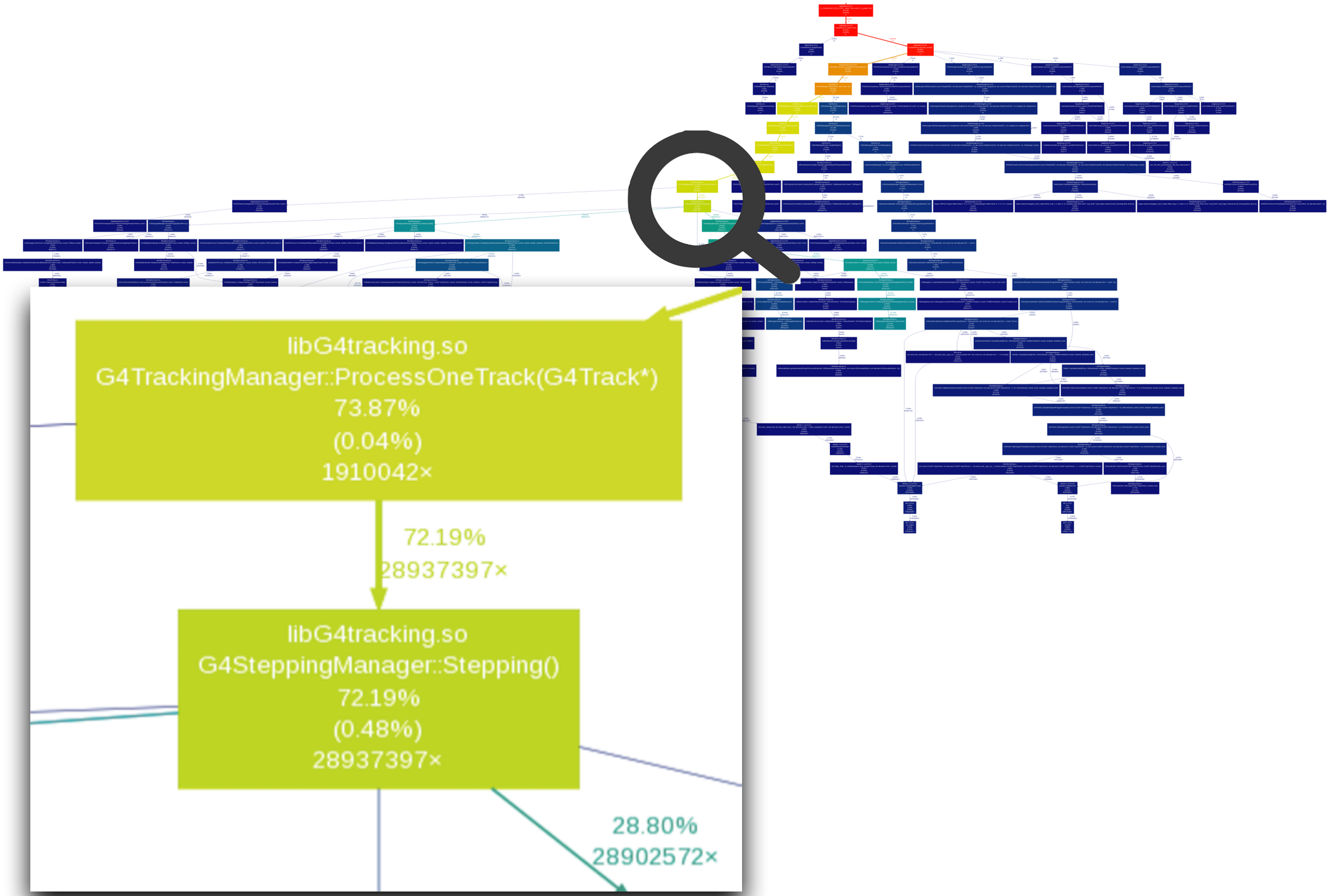
1 random mid-central HIJING 4-7 fm event, no absorber hits (memory limit on my laptop and my patience limit with callgrind). It includes shower generation, removal of redundant truth, full set of default evaluation, so tests both “reco" CPU and “analysis” CPU

highlighted positions in following slides

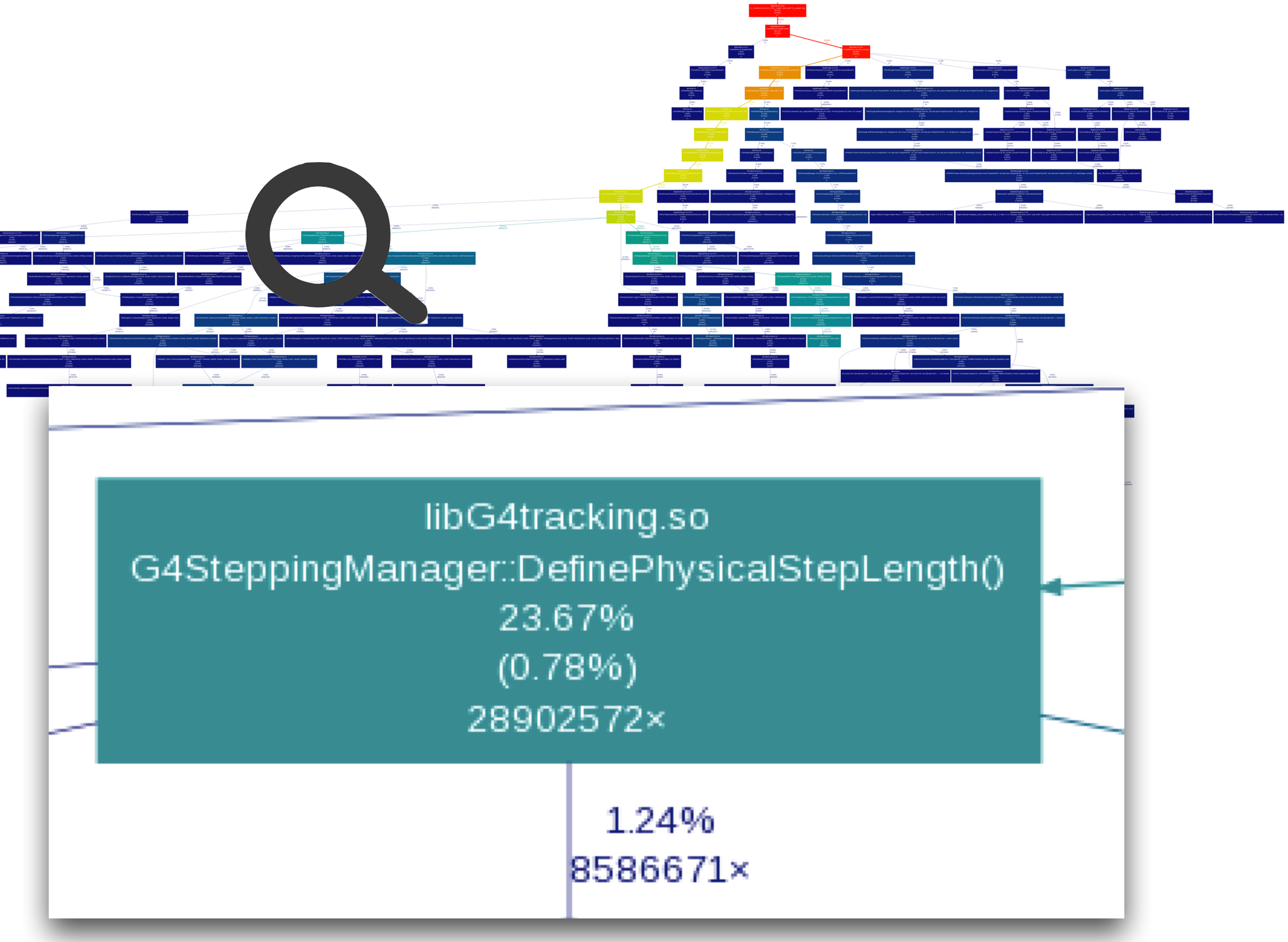
PROFILING MID-CENTRAL EVENTS



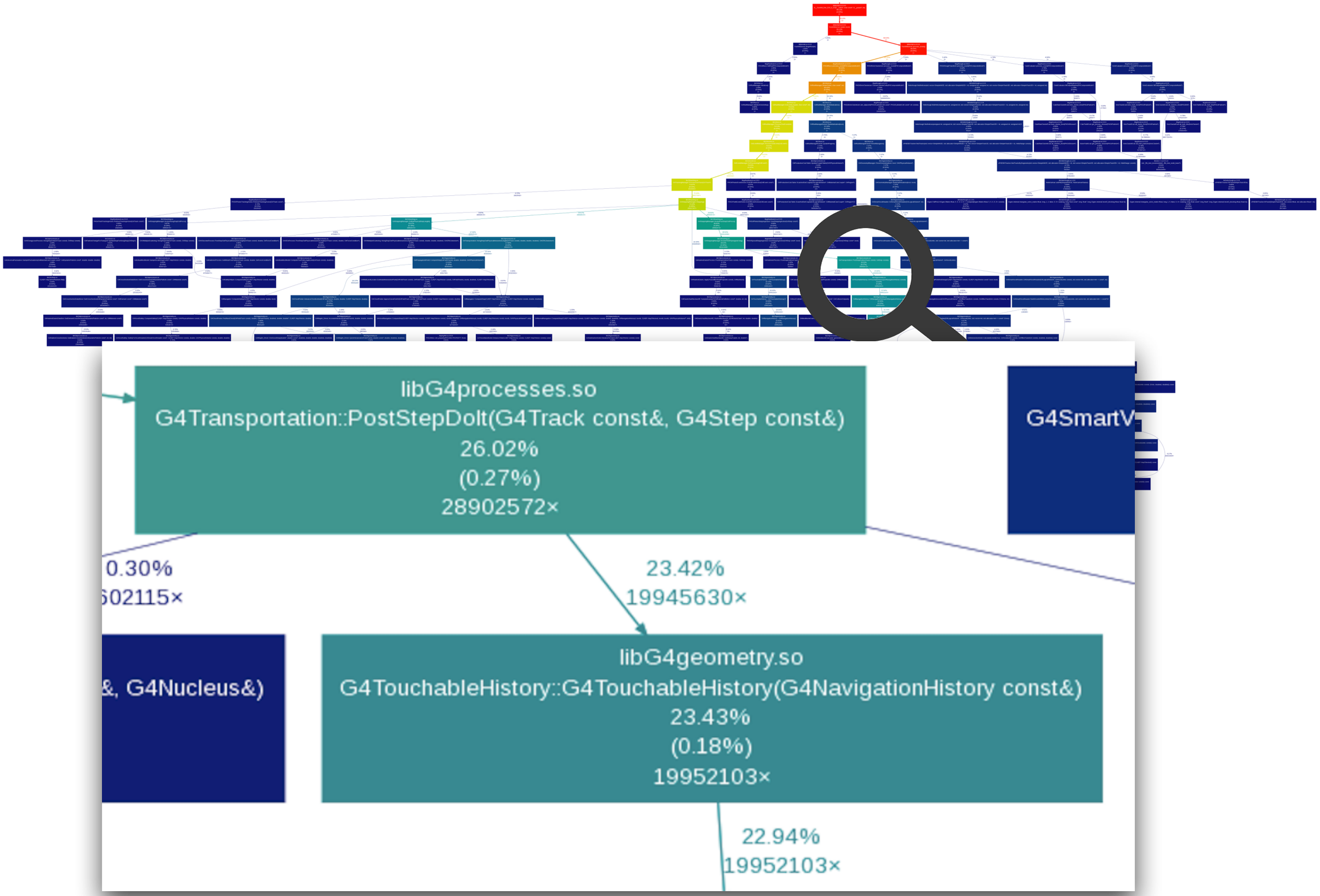
PROFILING MID-CENTRAL EVENTS



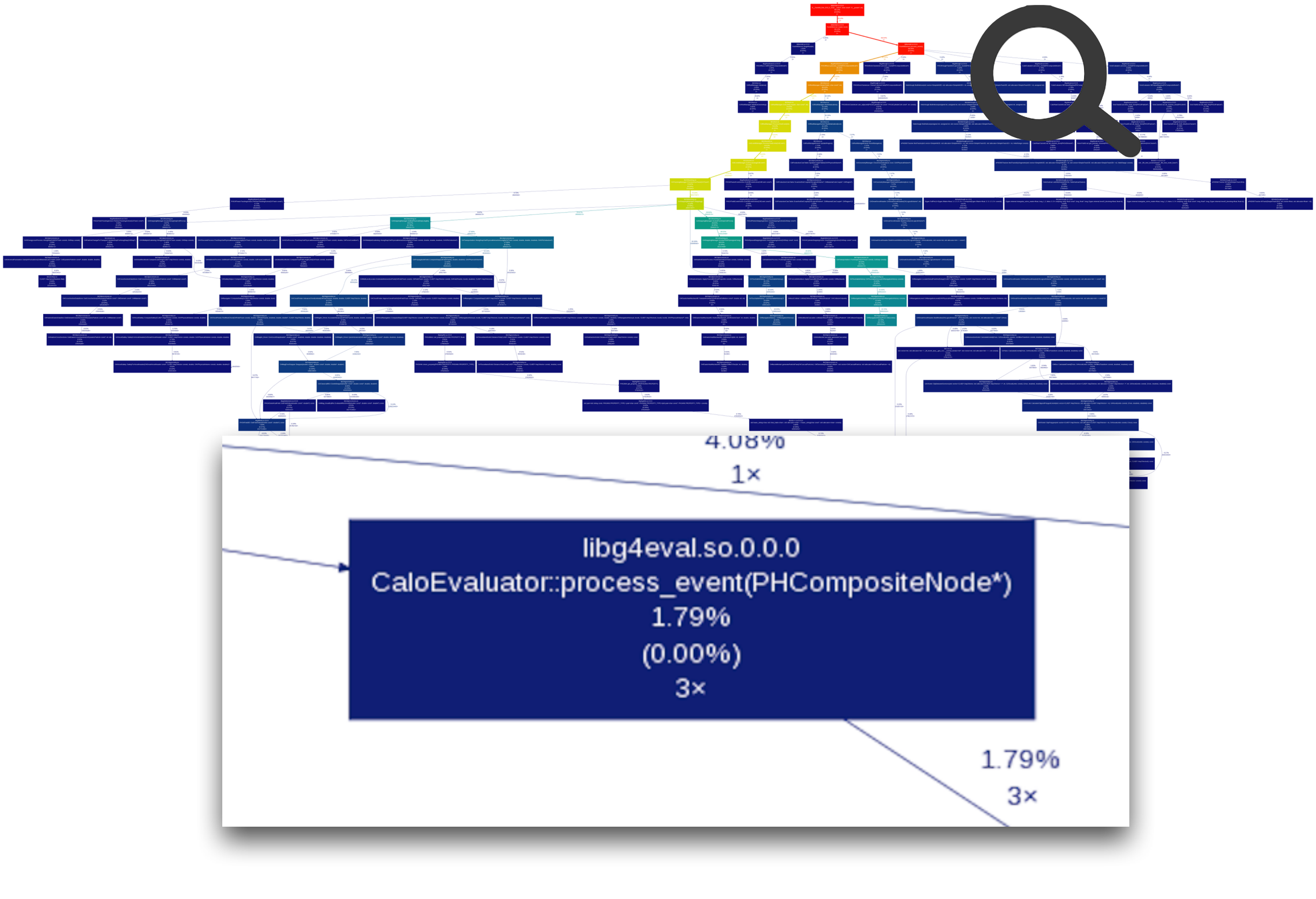
PROFILING MID-CENTRAL EVENTS



PROFILING MID-CENTRAL EVENTS



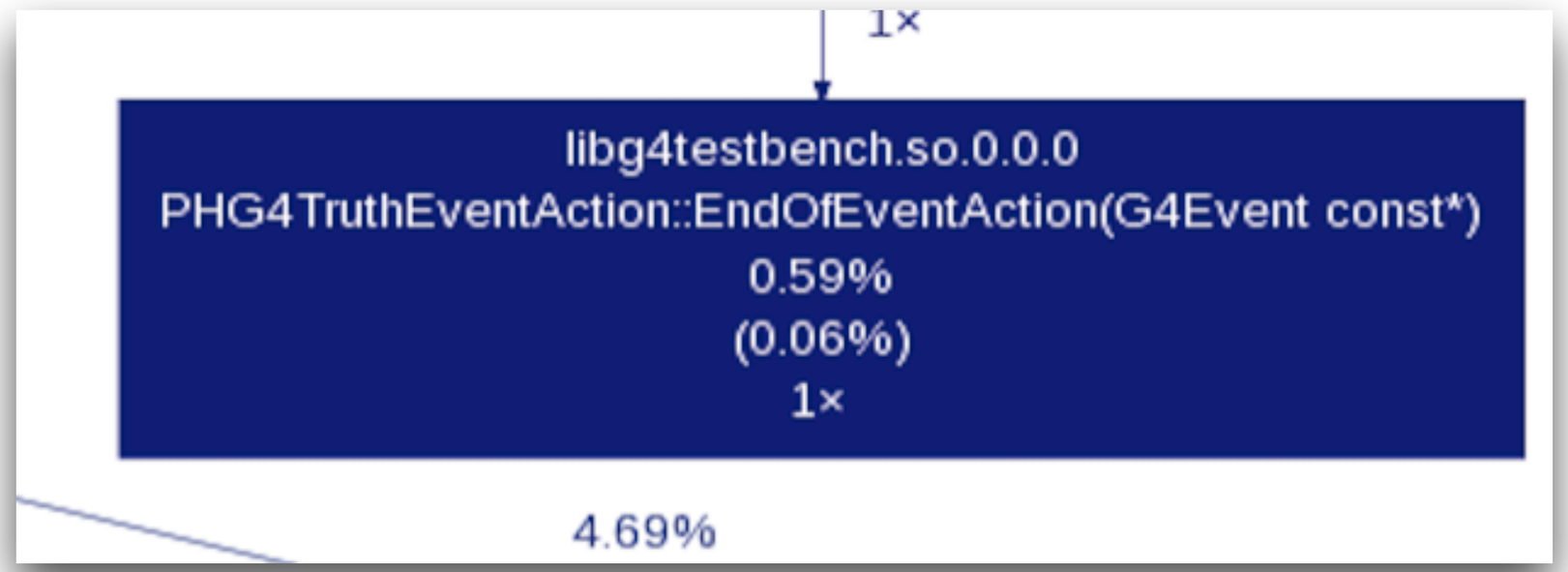
PROFILING MID-CENTRAL EVENTS



PROFILING MID-CENTRAL EVENTS

libg4testbench.so.0.0.0
PHG4TruthTrackingAction::PreUserTrackingAction(G4Track const*)
0.69%
(0.10%)
1910042×

PROFILING MID-CENTRAL EVENTS



SUMMARY

Changes are available in Pull Request #101. DST size performance and CPU performance are very encouraging. Most run-time errors are patched.

Implementation comments, criticisms are requested. How could this work differently?

I need feedback on what the DST format will look like: Do we write out empty nodes or remove them completely? so that I can fully test that read back

Jin wants sub-showers to manage radiative photons, these can be stored as redundant secondary showers. Map storage is ready for sub-shower storage. One option: write out every first generation sub shower. Other ideas are requested, but more complicated ideas need an implementation plan or at least a very specific algorithm for creation.

Backup Slides